L'héritage multiple

Le langage python permet la mise en place d'héritage multiple (une classe héritant de plusieurs classes).

```
class MereA:
    pass

class MereB:
    pass


class Enfant(MereA, MereB):
    pass
```

L'héritage multiple

Cependant, cela peut poser problème :

Lorsque plusieurs classes mères définissent la même méthode :

- Quel méthode sera utilisée par défaut (sans utiliser la redéfinition) ?
- Comment cibler la méthode d'un des parents au sein de la classe Enfant ?

C'est d'ailleurs pour ses raisons que peu de langages permettent la mise en place d'héritage multiple. Par exemple, les langages "C#" et "Java" ne le permettent pas.

L'héritage multiple

Pour éviter ses problèmes, en cas de conflit le python redéfinie les méthodes.

L'attribut de classe « __mro__ » permet d'obtenir un tuple avec l'ordre de redéfinition.

```
(<class '__main__.Enfant'>, <class '__main__.MereA'>, <class '__main__.MereB'>, <class 'object'>)
```

La méthode « super() » accède aux méthodes de la classe parent sur base de cet ordre.

Pour cibler la méthode d'une classe particulière, il est possible d'utiliser le nom de la classe à la place de la méthode « super() ».

Exemple d'héritage multiple

```python
class MereA:

    def __init__(self, val_a):
        self.val_a = val_a

    def methode(self):
        print("Elem A")


class MereB:

    def __init__(self, val_b):
        self.val_b = val_b

    def methode(self):
        print("Elem B")
```

```python
class Enfant(MereA, MereB):

    def __init__(self, val_a, val_b):
        MereA.__init__(self, val_a)
        MereB.__init__(self, val_b)

    def methode_1(self):
        super().methode()    # Elem A

    def methode_2(self):
        MereA.methode(self) # Elem A

    def methode_3(self):
        MereB.methode(self) # Elem B
```